Contents lists available at ScienceDirect

Computers in Human Behavior

journal homepage: www.elsevier.com/locate/comphumbeh

Review

Review on teaching and learning of computational thinking through programming: What is next for K-12?

Sze Yee Lye [*], Joyce Hwee Ling Koh

National Institute of Education, Nanyang Technological University, Singapore, 1 Nanyang Walk, Singapore 637616, Singapore

ARTICLE INFO

Article history:
Available online 30 September 2014

Keywords:
Programming
Scratch
Computer science education
K-12
Computational thinking

ABSTRACT

Programming is more than just coding, for, it exposes students to computational thinking which involves problem-solving using computer science concepts like abstraction and decomposition. Even for non-computing majors, computational thinking is applicable and useful in their daily lives. The three dimensions of computational thinking are computational concepts, computational practices and computational perspectives. In recent years, the availability of free and user-friendly programming languages has fuelled the interest of researchers and educators to explore how computational thinking can be introduced in K-12 contexts. Through an analysis of 27 available intervention studies, this paper presents the current trends of empirical research in the development of computational thinking through programming and suggests possible research and instructional implications. From the review, we propose that more K-12 intervention studies centering on computational practices and computational perspectives could be conducted in the regular classroom. To better examine these two dimensions, students could be asked to verbalize their thought process using think aloud protocol while programming and their on-screen programming activity could be captured and analyzed. Predetermined categories based on both past and recent programming studies could be used to guide the analysis of the qualitative data. As for the instructional implication, it is proposed that a constructionism-based problem-solving learning environment, with information processing, scaffolding and reflection activities, could be designed to foster computational practices and computational perspectives.

Contents

* Corresponding author.
E-mail addresses: lye.szeyee@gmail.com (S.Y. Lye), joyce.koh@nie.edu.sg (J.H.L. Koh).

http://dx.doi.org/10.1016/j.chb.2014.09.012
0747-5632/© 2014 Elsevier Ltd. All rights reserved.

1. Introduction

Programming for K-12 can be traced to the 1960s when Logo programming was first introduced as a potential framework for teaching mathematics (Feurzeig & Papert, 2011). In Logo, the students moves the turtle (arrow) on the screen by issuing commands like FD 100 (forward 100). In his seminal book "Mindstorms: Children, computers and powerful ideas", Papert (1980) advocated the use of the discovery constructionist mode for learning Logo. Nevertheless, Logo did not catch on in mainstream schools in the 1980s, possibly because of the incompatibility between its discovery-enabled approach and the more conventional behaviourist school culture back then (Agalianos, Noss, & Whitty, 2001). Papert (1980) claimed that the Logo programming experience could develop powerful intellectual thinking skills among children. Contrary to his claim, empirical studies of Logo programming did not find conclusive evidence of it improving the thinking skills of children (Kurland, Pea, Clement, & Mawby, 1986; Pea, 1983).

After Logo, the use of programming to teach thinking skills in K-12 was not extensively reported. However, in the recent years, there has been renewed interest in introducing programming to K-12 students (Grover & Pea, 2013; Kafai & Burke, 2013). This is fuelled by the availability of easy-to-use visual programming languages such as Scratch (Burke, 2012; Lee, 2010), Toontalk (Kahn, Sendova, Sacristán, & Noss, 2011), Stagecast Creator (Denner, Werner, & Ortiz, 2012) and Alice (Graczyńska, 2010). Many of these new programming languages such as Scratch and Alice have been modelled after aspects of Logo (Utting, Cooper, Kölling, Maloney, & Resnick, 2010).

During programming, students are exposed to computational thinking, a term popularized by Wing (2006). It involves the use of computer science concepts such as abstraction, debugging, remixing and iteration to solve problems (Brennan & Resnick, 2012; Ioannidou, Bennett, Repenning, Koh, & Basawapatna, 2011; Wing, 2008). This form of thinking can be considered to be fundamental for K-12 students because it requires "thinking at multiple abstractions" (Wing, 2006, p. 35). More importantly, computational thinking is in line with many aspects of 21st century competencies such as creativity, critical thinking, and problem- solving (Ananiadou & Claro, 2009; Binkley et al., 2012). Thus, it is not surprising that many educators assert that programming is important for K-12 students in this era (Kafai & Burke, 2013; Margolis, Goode, & Bernier, 2011; Resnick et al., 2009). This revived interest in programming for K-12 settings suggests the need to consider how it can be better related to the kinds of educational outcomes that it can potentially foster. Some of the outcomes suggested by researchers are the ability to think more systematically (Kafai & Burke, 2013) and the development of mathematical and scientific expertise (Sengupta, Kinnebrew, Basu, Biswas, & Clark, 2013). Yet, in the current literature, there is a dearth of papers that explore computational thinking through programming in K-12 contexts (Grover & Pea, 2013) as these programming studies are more often examined for tertiary students undertaking computer science courses (e.g., Katai & Toth, 2010; Moreno, 2012). Therefore, in this paper, we attempt to examine published empirical studies involving students in both K-12 and higher education contexts so as to derive insights on computational thinking through programming for K-12 curriculum.

2. Computational thinking

2.1. Definition

The term computational thinking is made popular by Wing (2006). In her seminal article on computational thinking, she argued that computational thinking "represents a universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use" (p. 33). Since then, computational thinking has gained traction in the K-12 context in the USA. However, the definition of computational thinking still remains contested as no dominant discourse reigns (Barr & Stephenson, 2011; Brennan & Resnick, 2012; Grover & Pea, 2013). For example, the International Society for Technology in Education (ISTE) views computational thinking as algorithmic thinking with automation tools and data representation with the use of simulation. On the other hand, the National Research Council (NRC) recommends mathematics and computational thinking to be one of the eight essential practices for the scientific and engineering dimension outlined in the "Framework for K-12 Science Education" (NRC, 2012). In this framework, mathematics and computational thinking involves the use of computer tools to represent physical variables and the relationships among them.

For both ISTE and NRC, students may be considered to be exhibiting computational thinking even though they are not creating with technology tools. Conversely, programming involves students exhibiting computational thinking through the construction of artifacts (Kafai & Burke, 2013; Resnick et al., 2009). Thus, the general definitions on computational thinking suggested by ISTE and NRC may not be suited for programming. Hence, in this review on computational thinking through programming for K-12 students, we are using the framework proposed for Scratch by Brennan and Resnick (2012). Scratch is a popular programming language used in K-12 settings (e.g., Baytak & Land, 2011; Kafai, Fields, & Burke, 2010; Tangney, Oldham, Conneely, Barrett, & Lawlor, 2010; Theodorou & Kordaki, 2010). With respect to Scratch, Brennan and Resnick (2012) proposed three dimensions of computational thinking: computational concepts, computational practices, and computational perspectives. Table 1 summarizes the key ideas on these three dimensions. These dimensions are appropriate for understanding how K-12 students approach programming as they are also in line with the Logo programming language knowledge proposed by Mayer (1992). This includes the syntactic, semantic, schematic knowledge (computational

Table 1
Computational thinking.

Dimension	Description	Examples
Computational concepts	Concepts that programmer use	Variables Loops
Computational practices	Problem-solving practices that occurs in the process of programming	Being incremental and iterative Testing and debugging Reusing and remixing Abstracting and modularizing
Computational perspectives	Students' understandings of themselves, their relationships to others, and the technological world around them	Expressing and questioning about the technology world

concepts) and strategic knowledge (computational practices). Moreover, Scratch shares similar features with contemporary visual programming languages for K-12 students (e.g., Alice). These languages are easy-to-understand as they, provide visual feedback of the program in the form of animated objects and allow students to create interactive media (e.g., animations and games). Therefore, this framework is likely to be suitable for considering computational thinking for programming contexts in K-12 education.

2.2. Computational thinking through K-12 programming tools

Traditional programming languages such as Java or C++ have representation that closely resembles the computer's way of thinking (Smith, Cypher, & Tesler, 2000). On the other hand, visual programming languages use representation that is closer to human language. These visual programming languages are usually less powerful than traditional languages as they are domain-specific (e.g., 3D animation for Alice). It is better to use visual programming languages rather than traditional programming languages to facilitate the three dimensions of computational thinking in K-12 contexts because unnecessary syntax is reduced (e.g., the use of semi colon and curly brackets) and the commands are closer to spoken English. Students usually need only to drag and snap the command blocks (see Fig. 1). With these features, such programming tools help reduce the cognitive load on the students and "allow students to focus on the logic and structures involved in programming rather than worrying about the mechanics of writing programs" (Kelleher & Pausch, 2005, p. 131). As such, these features of visual programming languages can potentially allow students to acquire the computational concepts more easily without the need to learn complex programming syntax.

These programming tools also facilitate students to enact the computational practices dimension of computational thinking more easily because the outcomes of their programming can be viewed in the form of animated objects. Such visualization makes computational practices such as testing and debugging cognitively less demanding. This allows students to acquire computational problem-solving practices more easily. Ultimately, these tools become "technology-as-partner in the learning process" (Jonassen, Howland, Marra, & Crismond, 2008, p. 7) and can possibly help K-12 students to extend these computational practices towards enhancing their general problem-solving ability (Lin & Liu, 2012; Ratcliff & Anderson, 2011). These tools can also engage students in the building of multi-media digital products, thereby enabling programming activities to be used as a means for students to express their ideas. This can shape students' computational perspective about the technological world. It develops students' digital literacy for creating, sharing and remixing digital resources (Hague & Payton, 2011; Mills, 2010; Ng, 2012) and in the process of doing so, students are no longer passive consumers of the technology (Resnick et al., 2009). K-12 programming tools are therefore becoming increasingly important because they afford for such kinds of digital literacy experiences (Mills, 2010).

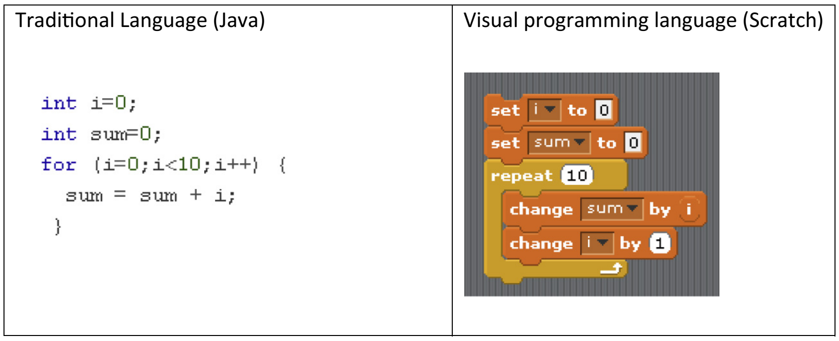
3. Research purpose

With these recent developments in the visual programming languages for K-12, there is renewed interest to consider how programming can benefit K-12 students (Barr & Stephenson, 2011; Bell, 2013; Grover & Pea, 2013; Olson, 2012). There is, clearly, a need for researchers and educators to better understand the empirical evidence from studies related to students' engagement in programming activities. Hence, the purpose of this paper is to suggest possible research and instructional implications based on the review of existing empirical studies. Specifically, this review is guided by the following questions:

1. How has programming been incorporated into K-12 curricula?
2. What are the reported outcomes of student performance in computational thinking dimensions?
3. What intervention approaches are being used to foster computational thinking?

4. Search procedures

Searches were performed for recent published peer-reviewed empirical intervention studies on computational thinking through programming. Therefore, the search excluded conference proceedings papers and conceptual papers. We chose to start the search from the year 2009 as it was the year where the NRC conducted the first workshop to discuss how best to introduce students to computational thinking in the USA. Furthermore, ISTE and the Computer Science Teachers' Association also started a project enti-



Traditional Language (Java)

```
int i=0;
int sum=0;
for (i=0;i<10;i++) {
  sum = sum + i;
 }
```

Visual programming language (Scratch)

```
set i to 0
set sum to 0
repeat 10
  change sum by i
  change i by 1
```

Fig. 1. Programming languages.

tled "Leveraging Thought Leadership for Computational Thinking in PK-12" in 2009.

We first started our search with the search term "computational thinking" and "K-12" in two popular and established databases: SSCI (Education educational research category) and ERIC. In ERIC, we only searched for peer reviewed articles. As of November 15 2013, the search returned 13 articles (2 from SSCI and 11 from ERIC). Due to the limited number of articles found, we decided to use just one search term "computational thinking". This search without the term "K-12" resulted in studies conducted in higher education context. These higher education studies, too, can inform K-12 studies as these interventions could be adapted for younger students. As of November 15 2013, the search returned 19 articles (2 from SSCI and 17 from ERIC) but none of the articles were selected since they were conceptual papers, literature reviews, or empirical studies where programming was not used to foster computational thinking. As the aim of this paper is to review articles on computational thinking through programming, we conducted another search in these two databases with the search term "computer programming". It yielded 109 articles (35 from SSCI and 74 from ERIC) where 20 articles were selected for review. The rest of the articles were discarded as they were either not empirical studies or not reporting on computational thinking. Due to the dearth of studies that have been conducted in this field, we decided to expand the search using a more general term of "computer science". It yielded 94 articles and only seven articles were selected. The other articles were discarded as their research focus was not programming. All in all, 27 articles were selected for this literature review.

5. Findings

5.1. Research question 1: How has programming been incorporated into K-12 curricula?

Out of the 27 studies reviewed, nine were carried out with K-12 students (see Table 2). The programming languages (e.g., Scratch and Logo) adopted for the younger students were typically "low-floor" (easy for the students to pick up) and "high-ceiling" (allow students to create more sophisticated programs) as envisioned by Papert (1980). The only exception was in the study of Wang and Chen (2010) where students were taught Flash ActionScript. Programming was used with a wide profile of K-12 students which included kindergarten and middle school students.

Students were found to be using programming to learn content such as languages or mathematics. In the learning of languages, Burke (2012) suggested that Scratch offered a "new medium through which children can exercise the composition skills they learned within traditional literacy classrooms while also offering the mutual benefit of introducing coding at earlier ages" (p. 131). Students with hearing disorders learnt English words with the use of Logo (Miller, 2009) whereas in mathematics, students externalized their mathematics concepts through Toontalk (Kahn et al., 2011) and Logo (Fessakis, Gouli, & Mavroudi, 2013). Students were also reported to be creating language art projects with Scratch (Lee, 2010).

5.2. Research question 2: What are the reported outcomes in terms student performance in the computational thinking dimensions?

From the review, most of the studies were exploring issues related to the computational thinking dimension of computational concepts. See Table 3 for details.

5.2.1. Computational concepts

Studies focusing on this computational thinking dimension examined how students learnt the technicalities of programming which included computational concepts such as variables and loops. There were altogether 23 studies (16 quantitative and seven qualitative). Seven studies were conducted in K-12 while the rest were conducted on higher education students who were learning more complex computational concepts (e.g., class and bubble sorting).

For the 16 quantitative studies, only two studies were conducted in K-12 settings. Both these studies reported results in favour of the experimental group with treatments such as onscreen blocks and game-play being used to teach computational concepts (Kazakoff & Bers, 2012; Wang & Chen, 2010). For the eight studies performed on higher education students, the results showed that students in the treatment group taught with strategies such as pair programming using metaphors, mindmapping and cooperative learning and multi-sensory methods performed better (Hui & Umar, 2011; Hung, 2012; Ismail, Ngah, & Umar, 2010; Katai & Toth, 2010; Kose, Koc, & Yucesoy, 2013; Kyungbin & Jonassen, 2011; Ma, Ferguson, Roper, & Wood, 2011; Moreno, 2012). Please refer to Table 3 for the details about the treatment for the experimental and control groups.

On the other hand, the remaining three higher education studies did not report any significant differences between the control and experimental groups. The non-significant results could be due to the experimental group spending less time on the programming tasks than the control group (Garner, 2009), short experimental time or small number of participants (Hsiao & Brusilovsky, 2011) and that the intervention did not include the learning of the test items (Urquiza-Fuentes & Velazquez-Iturbide, 2013). Despite the lack of significant differences between the treatment and the control group, these studies reported that the treatments had a positive effect on the weaker students. Strategies such

Table 2
K-12 studies.

Author	Participants	Profile	Age	Duration	Programming language	Subject learnt
Fessakis et al. (2013)	10	Kindergarten children	5–6	1 h 15 min	Logo	Mathematics
Burke (2012)	10	Male students in voluntary in after school program	12–14	7 weeks	Scratch	English
Denner et al. (2012)	59	Female students in voluntary after school program	12–14	14 months	Stagecast Creator	Computer programming
Kazakoff and Bers (2012)	58	Kindergarten children	4.5–6.5	20 h	Creative hybrid environment for robotic programming	Computer programming
Lin and Liu (2012)	3	Students in MSWLogo camp	9–10	5 days	Logo	Computer programming
Kahn et al. (2011)	31	High achieving students	9–13	7–10 weeks	Toontalk	Mathematics
Lee (2010)	1	After school program	9	6 months	Scratch	Language arts
Wang and Chen (2010)	115	Junior high students with flash experience	12–14	6 week	Flash action script	Computer programming
Miller (2009)	1	Hearing-impaired	13	3 months	Logo	English

Table 3
Summary of article.

	Author	Setting	Research approach	Intervention	Computational thinking		
					Concepts	Practices	Perspective
1	Fessakis et al. (2013)	K-12	Case study	Teacher-guided whole-class approach with interactive white board		✓	
2	Kose et al. (2013)	Higher education	Experimental	*Experimental* Story-based e-learning approach *Control* Traditional teacher-directed approach	✓		
3	Urquiza-Fuentes and Velazquez-Iturbide (2013)			*Experimental* Program visualization through animation construction or viewing *Control* Traditional teacher-directed approach	✓		
4	Burke (2012)	K-12	Case study	Digital story telling	✓		✓
5	Denner et al. (2012)		Qualitative artifact analysis	Game creation	✓		
6	Hung (2012)	Higher education	Experimental	*Experimental* Diagram method Analogy method *Control* Lecture style	✓		
7	Kazakoff and Bers (2012)	K-12	Experimental	*Experimental* Tangible programming language with the use of physical or onscreen blocks *Control* Art activities	✓		
8	Lin and Liu (2012)		Case study	Pair programming	✓	✓	
9	Moreno (2012)	Higher education	Experimental	*Experimental* Game strategy creation *Control* Regular exercise as homework with no interaction with the game	✓		
10	Wang et al., 2012		Case study	Peer code review	✓	✓	
11	Esteves et al. (2011)		Action research	Project based learning in virtual world		✓	
12	Hsiao and Brusilovsky (2011)		Experimental	*Experimental* Annotation and peer review of annotation *Control* Annotation and browsing through the annotation	✓		
13	Hui and Umar (2011)			*Experimental* Pair programming Metaphor *Control* Pair programming	✓		
14	Kahn et al. (2011)	K-12	Case study	Computational modelling of mathematics concepts Online discussion			✓
15	Kyungbin and Jonassen (2011)	Higher education	Experimental	*Experimental* Active reflective self explanation *Control* Passive reflective self explanation	✓	✓	
16	Ma et al. (2011)			*Experimental* Cognitive conflict Program construction *Control* Program construction	✓		
17	Moura and van Hattum-Janssen (2011)		Survey	Active learning	✓		
18	Robertson (2011)		Qualitative artifact analysis	Blogging		✓	
19	Goel and Kathuria (2010)		Experimental	*Experimental* Pair programming *Control* Solo programming	✓		
20	Wang and Chen (2010)	K-12		*Experimental* Game play using matching-challenging strategy *Control* Game play using challenging strategy	✓		

(*continued on next page*)

Table 3 (*continued*)

	Author	Setting	Research approach	Intervention	Computational thinking		
					Concepts	Practices	Perspective
21	Katai and Toth (2010)	Higher education		*Experimental*			
				Multi-sensory method such as dancing and role playing	✓		
				Control			
				Without multi-sensory method			
22	Garner (2009)			*Experimental*			
				Part-complete solution method	✓		
				Control			
				Without part-complete solution method			
23	Kordaki (2010)	Higher education	Case study	Computer-based problem-solving environment	✓		
24	Lee (2010)	K-12		Analogy-based instructional strategies			
				Individual project work	✓		
25	Ismail et al. (2010)	Higher education	Experimental	*Experimental*			
				Mindmapping	✓		
				Cooperative learning			
				Control			
				Traditional teacher-directed approach			
26	Jiau et al. (2009)			*Experimental*			
				Game strategy creation	✓		
				Control			
				Without game strategy creation			
27	Miller (2009)	K-12	Case study	Modelling	✓		

as having students to work on part-complete solutions (Garner, 2009) and reviewing comments on programming examples (Hsiao & Brusilovsky, 2011) were reported as being beneficial for the weaker students whereas another study reported that the failure and dropout rate decreased with the use of animation viewing and construction in a computer science course (Urquiza-Fuentes & Velazquez-Iturbide, 2013).

The remaining three quantitative studies only reported descriptive statistics which indicated that the intervention had helped the students. However, they did not analyze the pre and post-study statistical differences. Both the failure rate and withdrawal rate dropped in a computer science introductory course through the use of an learning-centred active approach (Moura & van Hattum-Janssen, 2011), weaker students caught up with the better students after experiencing pair programming (Goel & Kathuria, 2010) and the number of students scoring above 80 per-cent increased with game strategy creation (Jiau, Chen, & Ssu, 2009).

For the remaining seven non-experimental qualitative studies, students were found to have grasped the computational concepts based on field observation, survey, test results and artifact analysis. Unlike the quantitative studies, these studies described the students' programming experience. These studies were conducted in both K-12 (Burke, 2012; Denner et al., 2012; Lee, 2010; Lin & Liu, 2012; Miller, 2009) and higher education contexts (Kordaki, 2010; Wang, Li, Feng, Jiang, & Liu, 2012).

5.2.2. Computational practices

For the computational practices studies examining problem-solving processes during programming, Fessakis et al. (2013) and Esteves, Fonseca, Morgado, and Martins (2011) reported how the visualization output of the programming code helped the K-12 students and higher education students respectively in this dimension of computational thinking. In the study of Fessakis et al. (2013), kindergarten students were incremental and iterative while creating the paths with Logo as they were observed to prefer ''stepwise refinement approach which gave them the opportunity to immediately execute their commands and receive feedback'' (p. 94). On the other hand, the visualization of 3D output in the Second Life programming environment helped undergraduates in testing and debugging as ''the students had an obvious feedback regarding the correctness of their program'' (Esteves et al., 2011, p. 631).

The other four studies investigated how intervention approaches such as reflection affected students' computational practices. In these studies, only one was conducted for K-12 students (Lin & Liu, 2012). The results showed that the interventions had positive effect on the computational practices of being incremental and iterative (Robertson, 2011), and testing and debugging (Kyungbin & Jonassen, 2011; Lin & Liu, 2012; Wang et al., 2012).

5.2.3. Computational perspectives

Computational perspectives entail students developing understandings of themselves and their relationships with others and the technological world. For example, this dimension of computational thinking was evident when students were expressing themselves with programming. For the two studies that reported on computational perspectives, K-12 students were able to express themselves by creating interactive digital media using contemporary K-12 programming tools. High ability students expressed their concept of infinity (Kahn et al., 2011) with Toontalk while middle school students were able to create their own digital stories with Scratch (Burke, 2012).

5.3. Research question 3: What intervention approaches are being used to foster computational thinking?

To answer this research question, the interventions outlined in Table 3 were further analyzed and grouped into four categories. They are reinforcement of computational concepts, reflection, and information processing and constructing their own programs.

5.3.1. Reinforcement of computational concepts

In this review, the computational concepts were being reinforced with the help of the computer system where feedback was provided through game for junior high students (Wang & Chen, 2010) or e-learning approach for university students (Kose et al., 2013). These two studies reported positive results. This approach was grounded on behaviourist learning where the desired performance is shaped through the use of behaviour management strategies such as reinforcement and punishment (Driscoll, 2005). In such an approach, students are essentially learning from technology (technology-as-teacher). They are passive learners and information is transmitted to them. Such an approach is usually not favoured by contemporary researchers who advocate students learning with technology as partners in learning (e.g., Jonassen et al., 2008; Mayer, 2010).

5.3.2. Reflection

Reflection was a strategy more often used in the studies involving higher education students where they were asked to reflect on their programming experience. This can possibly foster computational practices and perspectives as the students need to review and think about their programming process. Such kinds of reflection can be directed towards their programming performance (Zimmerman & Tsikalas, 2005) or their peers' programming performance. Reflection was also found to encourage the review of one's own learning performance (Søndergaard & Mulder, 2012; Yang, 2010), thereby engaging the students into thinking-doing.

Most of the studies using this approach showed promising results. For self-reflection, participants in the study of Robertson (2011) constructed their own program and blogged about their programming experience in Second Life. It was found that the blogging experience could support the development of problem-solving practices such as being incremental and iterative for university students. On the other hand, engaging in self-explanation (Kyungbin & Jonassen, 2011) and peer code review (Wang et al., 2012) could help the students to test and debug. In the study of Hsiao and Brusilovsky (2011), the students reviewed the annotations on programming examples provided by the instructors. Even though this study did not report any findings on computational practices or perspectives, we surmise such peer review can possibly enhance these two dimensions of computational thinking

5.3.3. Information processing

The information processing approach helped students to acquire computational concepts by providing structures to allow them to better process the information presented to them. From the analysis, such approach was only evident in the higher education studies. This approach arises mainly from cognitivism where learning is viewed as the "processing of information and storing it in the memory" (Driscoll, 2005, p. 110). Most of the studies reported results in favour of the treatment group with the exception of Garner (2009) and Urquiza-Fuentes and Velazquez-Iturbide (2013).

In this review, researchers were using different strategies to enhance students' information processing. Metaphors were used to help students relate the programming concepts to their prior knowledge (Hui & Umar, 2011) and part-complete solutions helped to reduce students' cognitive load (Garner, 2009). Students were also asked to organize their thinking process by using mindmapping (Ismail et al., 2010) and program visualizations of the intermediate steps of their program (Ma et al., 2011; Urquiza-Fuentes & Velazquez-Iturbide, 2013). Another study used cognitive conflict to address students' misconceptions (Ma et al., 2011). In the study of Hung (2012), students' learning style were matched to the appropriate cognitive learning strategies (i.e., diagram and analogy method). Katai and Toth (2010) used a multi-sensory approach which affords for dual coding. In this study, students watched a dance performance and role-played so that they could better understand difficult computing concepts such as bubble-sorting.

In three studies (Hui & Umar, 2011; Ismail et al., 2010; Ma et al., 2011), these information processing intervention was extended by asking students to construct their own programs. As such, the studies did not just investigate how the learners acquired computational concepts in their head but learnt how to use the various computational concepts to build a workable program.

5.3.4. Constructing programs with scaffold

The most popular intervention approach involves learners constructing their own programs with scaffolds. Such approach could cover all the three dimensions of computational thinking but none of the studies examined the all the three aspects of computational thinking. There were altogether eight K-12 studies and ten higher education studies using this approach. The results of the studies suggested constructing programs could potentially help to foster the three dimensions of computational thinking. This approach arose from constructionism which "attaches special importance to the role of constructions in the world as a support for those in the head, thereby becoming less of a purely mentalist doctrine" (Papert, 1994, p. 143). In constructionism, students are actively engaged in knowledge construction by building meaningful products for others or themselves (Kafai & Resnick, 1996). For these studies, students constructed their programs with the scaffolds provided. They were not left alone to explore programming. This observation was in line with the assessment on Logo programming studies made by Mayer (2004) who argued that "the failure of pure discovery as an effective instructional method" (p. 17) for the learning of programming. Similarly, other Logo researchers too concurred with the view that structured guided discovery was a more preferred approach (Clement & Merriman, 1988; Lehrer, Lee, & Jeong, 1999).

For some of the studies, students were guided in their program construction with intervention approaches suggested in the previous section. These included reflection (Robertson, 2011) and information processing (Hui & Umar, 2011; Ismail et al., 2010; Ma et al., 2011). Other than these intervention approaches, the learners were also guided in their construction of their programs through computer scaffolding for program construction (Jiau et al., 2009; Kordaki, 2010; Ma et al., 2011; Moreno, 2012), teachers' scaffolding (Burke, 2012; Esteves et al., 2011; Fessakis et al., 2013; Kahn et al., 2011; Kazakoff & Bers, 2012; Lee, 2010; Miller, 2009; Moura & van Hattum-Janssen, 2011), guidance from parents (Lin & Liu, 2012) or learning from peers (Denner et al., 2012; Goel & Kathuria, 2010; Hui & Umar, 2011; Ismail et al., 2010). In particular, Goel and Kathuria (2010) and Hui and Umar (2011) adopted the strategy of pair programming. In pair programming, one student is the driver who does the actual coding while the other takes the role of an observer who reviews the code.

6. Research implications

6.1. Explore more classroom-based interventions

From Table 3, only nine peer-reviewed intervention studies were based in K-12 settings. There is thus an apparent gap in this research area of developing computational thinking (especially in the two dimensions of computational practices and computational perspectives) for K-12 students. Even with these limited studies, most were conducted as after-school activities. These students either participated in the activities voluntarily (e.g., Burke, 2012; Denner et al., 2012; Kahn et al., 2011; Lin & Liu, 2012) or were specially selected (e.g., Lee, 2010; Miller, 2009). Therefore, these studies might not be representative of typical classrooms and these results show that students' learning of computational thinking in naturalistic classrooms settings are still not well-understood. With the paucity of research in naturalistic classroom settings, there is, clearly, a need to conduct studies in this area. This will better help to inform educators and researchers on how to design and implement the grade-appropriate programming into K-12 curriculum.

6.2. Explore more studies in computational practices and computational perspectives

All in all, there were 23 (85%) studies that examined the learning outcomes in terms of computational concepts (e.g., conditions and variables). Computational thinking entails more than just the computational concepts. It also involves computational practices and perspectives as suggested by Brennan and Resnick (2012). However, there were only eight studies reporting either computational practices or computational perspectives. Studies examining computational practices and computational perspectives have become even more pertinent in K-12 settings as the rationale of introducing computational thinking (e.g., computational practices and computational perspectives) through programming is to equip them with the problem-solving skills that they can transfer to non-programming domains (Barr & Stephenson, 2011; Resnick et al., 2009; Wing, 2006). After all, computational practices and perspectives are especially useful in daily lives as there is methodical continuity between common sense and these two dimensions. Thus, the interventions for fostering of computational practices and computational perspectives, as well as the transfer of these competencies for general problem-solving is another area for further research to support the integration of programming into K-12 curricula.

6.3. Examining the programming process

For studies involving computational practices and computational perspectives, the programming process is usually examined. In this review, the programming process was mostly captured through field observations. But, such field observations usually involve a few participants and does not provide broad coverage (Yin, 2014). In future research, field observations could be further complemented by recordings of on-screen activities as demonstrated in the study conducted by Kahn et al. (2011). Furthermore, to better understand the students' programming process, the students can think-aloud while they are constructing their program. Such think-aloud protocol allows the cognitive process to be verbalized (Ericsson & Simon, 1993) and would provide useful information on the computational practice and perspectives. However, none of the articles reviewed adopted the think-aloud protocol. For future studies, researchers could consider getting the students to think-aloud while they are programming. Their on-screen activity with the verbalization of their thinking process would be recorded and analysed to better understand the computational practices and computational perspectives of computational thinking.

6.4. Analyzing qualitative data

In these eight studies, researchers were mostly collecting qualitative data such as field observations, interviews and students' artifacts to understand their computational practices and perspectives. These studies adopted conventional content analysis with no pre-determined categories to analyze the data. Conventional content analysis is suited for studies describing the phenomenon with scant literature (Hsieh & Shannon, 2005). But in this case, there are programming-related studies to guide the formation of categories since programming for K-12 can be traced to the 1960s with the introduction of Logo. Thus, for future research, these predetermined categories could be possibly be based on both past and recent programming studies (e.g., Ching & Kafai, 2008; Clement & Merriman, 1988; Klahr & Carver, 1988; Pea, 1983; Peppler & Kafai, 2007) or even seminar articles on problem-solving (e.g., Polya, 1957).

Table 4
Proposed debugging coding scheme.

Categories based on Polya (1957)	Possible codes
Understand the problem	Describe the bug Syntactic and semantic knowledge Program comprehension Causal reasoning
Devise a plan	Find the location bug Connection between the programming scripts Causal reasoning
Carry out the plan	Fix the bug
Review the plan	Test the solution

For example, for researchers interested in the computational practice of testing and debugging which is one of the essential problem-solving skill in programming (Fitzgerald et al., 2008; McCauley et al., 2008), the pre-determined categories could be derived from the generic established Polya (1957)'s 4-step of problem-solving. To further distill Polya's global strategy for use in testing and debugging, the actual set of possible codes are can be derived from for debugging Logo model proposed by Carver (1988). These include describing, finding, fixing the bug and testing the solution. Other possible codes also include the possible debugging skills such as connection between the programming scripts (Lehrer et al., 1999) and program comprehension (McCauley et al., 2008). This would provide the basis for the coding scheme in analyzing the qualitative data. Please see Table 4 for more details.

The use of such kinds of pre-determined categories will help researchers overcome the weakness of conventional content analysis in which they may not be able to grasp a "complete understanding of the context, thus failing to identify key categories" (Hsieh & Shannon, 2005, p. 1280).

7. Instructional implications for K-12

In the already limited number of studies on computational practice and computational perspectives, some researchers assumed that either the affordances of the visual programming environment were adequate to support students (Esteves et al., 2011; Fessakis et al., 2013) or that the students had the required abilities to undertake the learning tasks (Kahn et al., 2011). In essence, there was no specific intervention approach that considered both aspects during instruction. There seems to be an implicit assumption that learners can exhibit such computational practices and perspectives through pure self- discovery. However, we are of the view that this assumption needs to be interrogated as "the child's present experience is not self-explanatory" (Dewey, 1902/2008, p. 13). Without guidance on the cognitive aspects of computational practices and computational perspectives (Grover & Pea, 2013), the programming experience may be non-educative as students are not actively reflecting on their experience. They could be merely doing it in the trial-and-error mode rather than thinking as they are doing (Biesta & Burbules, 2003). Hence, when planning for programming in K-12 contexts, care needs to be devoted to these two aspects for supporting computational thinking. In essence, the students ought to be thinking-doing and not just doing.

To address the gap of scant intervention studies in computational practices and computational perspectives (see Section 6.2), there is a need to explore how instructional activities can support thinking-doing. We propose that researchers should consider designing K-12 constructionism-based problem-solving learning environment (PSLE) with evidence-based approaches as suggested by this review. Intervention approaches that are based on constructionism learning theory are common in the review of the 27 articles and these studies usually report positive outcomes. This PSLE could be designed with the framework suggested by Jonassen (2011) with activities planned for the intentional learning of problem-solving strategies. Students would learn to solve problems which are presented as cases and acquire cognitive skills such as causal reasoning and metacognition. The learning of such cognitive skills, though important for computational practices and computational perspectives, is rarely examined in the studies examined in this review.

In this PSLE, we envision students to be constructing program for an authentic situation (e.g., designing an interactive story for the school open house). The PSLE would also present students with instructional content (e.g., tutorial videos) for the computational concepts and cases they need to solve. The cases could contain bugs commonly generated by students during programming. The following would section further describe the design of the proposed PSLE.

7.1. Authentic problem

At the heart of this learning environment, there has to be a problem pertinent to the students since learning in problem-solving "should be anchored in an authentic problem that is relevant to the learner" (Jonassen, 2011, p. 150).They should be constructing things, in this case, programs, that matter to them. Hence, it is envisioned that they are more likely to be intellectually engaged (Kafai & Resnick, 1996). From Table 3, some of the possible problems as suggested by the review were designing game strategy (Jiau et al., 2009; Moreno, 2012), game (Denner & Werner, 2007; Lee, 2010) or digital stories (Burke, 2012; Lee, 2010). Researchers would have to contextualize the problems to their context.

7.2. Information processing activities

For students to acquire computational concepts, researchers should consider using the information processing approach (see Section 5.3.3) which is seldom used in the K-12 settings. Researchers may assume that students can acquire such computational concepts easily with the help of graphical and easy-to-use K-12 programming tools. However, despite the affordances of the K-12 programming tools, we argue that there is still the need for specific intervention approach for more complex concepts such as events which can cause another sub-program to execute. To help students better grasp such complex computing concepts, researchers can adapt information processing strategies (e.g., metaphor, cognitive conflict or mind-mapping) as suggested in this review.

7.3. Scaffolding process

The teacher would provide scaffolds for the students in this PSLE as suggested by the review (see Section 5.3.4). However, we find no framework on guiding the scaffolding process in these studies. We, thus, propose using the scaffolding process guided by the recommendations of Wood, Bruner, and Ross (1976). Please see Table 5.

Table 5
Role of teacher in scaffolding process (Wood et al., 1976).

Role	Description
Recruitment	Maintain the students' interest in the given task
Reduction in degrees of freedom	Making the task manageable
Direct maintenance	Keeping the students on task
Marking critical features	Highlighting feature that can help the students to accomplish the task
Frustration control	Motivate the students and provide timely guidance so that they would not feel frustrated and would like to give up
Demonstration	Role model the process required

To scaffold the program construction, the final program could be broken down into mini programs which would make the given task manageable (reduction in degrees of freedom). Take for example, in creating an interactive story, the students would need to know how to insert background or make the two objects to talk to each other. Teacher could demonstrate on how such a mini-program could be constructed. In frustration control, the teacher would guide the students by prompting them with questions on their problem-solving process (e.g., why do you put that command there? What is the purpose of this command?). Such "inserted questions may well be the most effective metacognitive strategy in problem-solving learning" (Jonassen, 2011, p. 170).

In demonstration, the teachers could present the case in the form of worked example which is one of the common instructional in problem-solving (Jonassen, 2011). During the worked example, the teachers could role model the problem-solving process by making explicit their thinking process. Such strategy is not new in programming studies. In the past Logo studies, there were explicit modelling of computational practice of abstracting and modularizing (Fay & Mayer, 1994) and testing and debugging (Craver, 1988).

Marking and emphasizing the critical features of causal relationship between the commands, especially in the different objects, is vital as this would help the students in program comprehension. For example, an event in one object might cause the execution of the command in another object. Program comprehension aids in computational practice such as testing and debugging (McCauley et al., 2008). But, novice programmers usually have difficulty relating different commands together (Robins, Rountree, & Rountree, 2003) as they would "identify programming actions at the level of individual programming statements" (Lehrer et al., 1999, p. 247). Hence, such explicit marking of causal reasoning is important for them to better understand the program and can thus better test and debug the program.

7.4. Reflection

On top of solving cases and constructing programs, the students could also be engaged in reflection which is seldom found in the K-12 studies reviewed. Reflection will help the students in metacognition as "reflection can focus on goals or one's own thinking" (Davis, 2003, p. 92). K-12 researchers can still consider adopting this approach so that the younger students reflect on their computational thinking process too (see Section 5.3.2). The students could either self-reflect on their own learning experience or reflect on their peers' code. Students would need to be guided on how they are engaged in self-reflection or peer reviewing. This guide could be adapted from Polya's problem-solving process.

8. Conclusion

In this paper, 27 empirical articles on programming in K-12 and higher education were reviewed. K-12 students were using easy to use visual programming languages to create digital stories and games. The popular intervention strategy is based on constructionism in which students create something concrete (e.g. program or comments) to consolidate what they have learned. Most of the studies reported positive outcomes. Despite the recent revived interest in programming for K-12, little studies have been conducted to inform the researchers and educators on implementing suitable curriculum for the group of students. In this paper, we recommend that more intervention studies, centering on computational practices and perspectives, can be conducted in regular K-12 classrooms. Rich data on computational practice and perspective could be collected via on-screen recording and students' thinking aloud while data analysis could be further strengthened by using predetermined categories based on programming studies. To support these two dimensions of computational thinking, a constructionism-based problem-solving learning environment, with authentic problem, information processing, scaffolding and reflection activities, could possibly be designed.

References

Agalianos, A., Noss, R., & Whitty, G. (2001). Logo in mainstream schools: The struggle over the soul of an educational innovation. *British Journal of Sociology of Education, 22*(4), 479–500.

Ananiadou, K., & Claro, M. (2009). 21st Century skills and competences for new millennium learners in OECD Countries. *OECD Education Working Papers*, 41.

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: What is Involved and What is the role of the computer science education community? *ACM Inroads, 2*(1), 48–54.

Baytak, A., & Land, S. M. (2011). An investigation of the artifacts and process of constructing computers games about environmental science in a fifth grade classroom. *Etr&D-Educational Technology Research and Development, 59*(6), 765–782.

Bell, S. (2013). Programming ability is the new digital divide: Berners-Lee. In *Computerworld*.

Biesta, G. J. J., & Burbules, N. C. (2003). *Pragmatism and educational research*. Lanham, MD: Rowman & Littlefield.

Binkley, M., Erstad, O., Herman, J., Raizen, S., Ripley, M., Miller-Ricci, M., et al. (2012). Defining twenty-first century skills. In P. Griffin, B. McGaw, & E. Care (Eds.), *Assessment and teaching of 21st century skills* (pp. 17–66). Netherlands: Springer.

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In *Annual American Educational Research Association meeting*, Vancouver, BC, Canada.

Burke, Q. (2012). The markings of a new pencil: Introducing programming-as-writing in the middle school classroom. *Journal of Media Literacy Education, 4*(2), 121–135.

Carver, S. (1988). Learning and transfer of debugging skills: Applying task analysis to curriculum design and assessment. In R. E. Mayer (Ed.), *Teaching and learning computer programming* (pp. 259–298). Hillsdale, NJ: Erlbaum.

Ching, C. C., & Kafai, Y. (2008). Peer pedagogy: Student collaboration and reflection in a learning-through-design project. *Teachers College Record, 110*(12), 2601–2632.

Clement, D. H., & Merriman, S. (1988). Componential developments in Logo programming and environments. In R. E. Mayer (Ed.), *Teaching and learning computer programming* (pp. 13–54). Hillsdale, NJ: Erlbaum.

Craver, S. M. (1988). Learning and transfer of debugging skills: Applying task analysis to curriculum design and assessment. In R. E. Mayer (Ed.), *Teaching and learning computer programming* (pp. 259–298). Hillsdale, NJ: Erlbaum.

Davis, E. A. (2003). Prompting middle school science students for productive reflection: generic and directed prompts. *Journal of the Learning Sciences, 12*(1), 91–142.

Denner, J., & Werner, L. (2007). Computer programming in middle School: How pairs respond to challenges. *Journal of Educational Computing Research, 37*(2), 131–150.

Denner, J., Werner, L., & Ortiz, E. (2012). Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts? *Computers & Education, 58*(1), 240–249.

Dewey, J. (1902/2008). *The child and the curriculum including, the school and society*. In. New York: Cosimo.

Driscoll, M. P. (2005). *Psychology of learning for instruction*. Boston, MA: Ally and Bacon.

Ericsson, K. A., & Simon, H. A. (1993). *Protocol analysis: Verbal reports as data* (2nd ed.). Boston: MIT Press.

Esteves, M., Fonseca, B., Morgado, L., & Martins, P. (2011). Improving teaching and learning of computer programming through the use of the Second Life virtual world. *British Journal of Educational Technology, 42*(4), 624–637.

Fay, A. L., & Mayer, R. E. (1994). Benefits of teaching design skills before teaching LOGO computer programming: Evidence for syntax-independent learning. *Journal of Educational Computing Research, 11*(3), 187–210.

Fessakis, G., Gouli, E., & Mavroudi, E. (2013). Problem solving by 5–6 years old kindergarten children in a computer programming environment: A case study. *Computers & Education, 63*, 87–97.

Feurzeig, W., & Papert, S. A. (2011). Programming-languages as a conceptual framework for teaching mathematics. *Interactive Learning Environments, 19*(5), 487–501.

Fitzgerald, S., Lewandowski, G., McCauley, R., Murphy, L., Simon, B., Thomas, L., et al. (2008). Debugging: Finding, fixing and flailing, a multi-institutional study of novice debuggers. *Computer Science Education, 18*(2), 93–116.

Garner, S. (2009). A quantitative study of a software tool that supports a part-complete solution method on learning outcomes. *Journal of Information Technology Education, 8*, 285–310.

Goel, S., & Kathuria, V. (2010). A novel approach for collaborative pair programming. *Journal of Information Technology Education, 9*, 183–196.

Graczyńska, E. (2010). ALICE as a tool for programming at schools. *Natural Science, 2*(2), 124–129.

Grover, S., & Pea, R. (2013). Computational thinking in K-12: A review of the state of the field. *Educational Researcher, 42*(1), 38–43.

Hague, C., & Payton, S. (2011). Digital literacy across the curriculum. *Curriculum Leadership, 9*(10).

Hsiao, I. H., & Brusilovsky, P. (2011). The role of community feedback in the student example authoring process: An evaluation of AnnotEx. *British Journal of Educational Technology, 42*(3), 482–499.

Hsieh, H. F., & Shannon, S. E. (2005). Three approaches to qualitative content analysis. *Qualitative Health Research, 15*(9), 1277–1288.

Hui, T. H., & Umar, I. N. (2011). Does a combination of metaphor and pairng activity help programming performances of students with different self-regulated learning level? *The Turkish Online Journal of Educational Technology, 10*(4), 122–129.

Hung, Y.-C. (2012). The effect of teaching methods and learning style on learning program design in web-based education systems. *Journal of Educational Computing Research, 47*(4), 409–427.

Ioannidou, A., Bennett, V., Repenning, A., Koh, K. H., & Basawapatna, A. (2011). Computational thinking pattern. In *Annual American Educational Research Association meeting*. New Orleans, Louisiana, United States.

Ismail, M. N., Ngah, N. A., & Umar, I. N. (2010). The effects of mind mapping with cooperative learning on programing performance, problem solving skill and metacognitive knowledge among computer science students. *Journal of Educational Computing Research, 42*(1), 35–61.

Jiau, H. C., Chen, J. C., & Ssu, K.-F. (2009). Enhancing self-motivation in learning programming using game-based simulation and metrics. *IEEE Transactions on Education, 52*(4), 555–562.

Jonassen, D. (2011). *Learning to solve problems: A handbook for designing problem-solving learning environments*. New York: Routledge.

Jonassen, D., Howland, J., Marra, R.M., & Crismond, D. (2008). *Meaningful learning with technology* (3rd ed.): Pearson/Merrill Prentice Hall.

Kafai, Y., & Resnick, M. (1996). *Constructionism in practice: Designing, thinking, and learning in a digital world*. In Lawrence Erlbaum.

Kafai, Y., & Burke, Q. (2013). Computer programming goes back to school. *Phi Delta Kappan, 95*(1), 61–65.

Kafai, Y., Fields, D. A., & Burke, Q. (2010). Entering the clubhouse: Case studies of young programmers joining the online Scratch communities. *Journal of Organizational and End User Computing, 22*(2), 21–35.

Kahn, K., Sendova, E., Sacristán, A. I., & Noss, R. (2011). Young students exploring cardinality by constructing infinite processes. *Technology, Knowledge and Learning, 16*(1), 3–34.

Katai, Z., & Toth, L. (2010). Technologically and artistically enhanced multi-sensory computer-programming education. *Teaching and Teacher Education, 26*(2), 244–251.

Kazakoff, E., & Bers, M. (2012). Programming in a robotics context in the kindergarten classroom: The impact on sequencing skills. *Journal of Educational Multimedia and Hypermedia, 21*(4), 371–391.

Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys, 37*(2), 83–137.

Klahr, D., & Carver, S. M. (1988). Cognitive objectives in a LOGO debugging curriculum: Instruction, learning, and transfer. *Cognitive Psychology, 20*(3), 362–404.

Kordaki, M. (2010). A drawing and multi-representational computer environment for beginners' learning of programming using C: Design and pilot formative evaluation. *Computers & Education, 54*(1), 69–87.

Kose, U., Koc, D., & Yucesoy, S. A. (2013). Design and development of a sample "computer programming" course tool via story-based e-learning approach. *Kuram Ve Uygulamada Egitim Bilimleri, 13*(2), 1235–1250.

Kurland, D. M., Pea, R., Clement, C., & Mawby, R. (1986). A study of the development of programming ability and thinking skills in high school students. *Journal of Educational Computing Research, 2*(4), 429–458.

Kyungbin, K., & Jonassen, D. H. (2011). The influence of reflective self-explanations on problem-solving performance. *Journal of Educational Computing Research, 44*(3), 247–263.

Lee, Y.-J. (2010). Developing computer programming concepts and skills via technology-enriched language-art projects: A case study. *Journal of Educational Multimedia and Hypermedia, 19*(3), 307–326.

Lehrer, R., Lee, M., & Jeong, A. (1999). Reflective teaching of Logo. *Journal of the Learning Sciences, 8*(2), 245–289.

Lin, J. M. C., & Liu, S. F. (2012). An investigation into parent-child collaboration in learning computer programming. *Educational Technology & Society, 15*(1), 162–173.

Ma, L., Ferguson, J., Roper, M., & Wood, M. (2011). Investigating and improving the models of programming concepts held by novice programmers. *Computer Science Education, 21*(1), 57–80.

Margolis, J., Goode, J., & Bernier, D. (2011). The need for computer science. *Educational Leadership, 68*(5), 68–72.

Mayer, R. E (1992). Teaching for transfer of problem-solving skills to computer programming. In E. Corte, M. Linn, H. Mandl, & L. Verschaffel (Eds.). *Computer-*

based learning environments and problem solving (Vol. 84, pp. 193–206). Berlin, Heidelberg: Springer.
Mayer, R. E. (2004). Should there be a three-strikes rule against pure discovery learning? *American Psychologist, 59*(1), 14–19.
Mayer, R. E. (2010). Learning with technology. In H. Dumont, D. Istance & F. Benavides (Eds.), *Nature of learning: Using research to inspire practice*. Paris, FRA: Organisation for Economic Cooperation and Development (OECD).
McCauley, R., Fitzgerald, S., Lewandowski, G., Murphy, L., Simon, B., Thomas, L., et al. (2008). Debugging: A review of the literature from an educational perspective. *Computer Science Education, 18*(2), 67–92.
Miller, P. (2009). Learning with a missing dense: What can we learn from the interaction of a deaf child with a turtle? *American Annals of the Deaf, 154*(1), 71–82.
Mills, K. A. (2010). A review of the "digital turn" in the new literacy studies. *Review of Educational Research, 80*(2), 246–271.
Moreno, J. (2012). Digital competition game to improve programming skills. *Educational Technology & Society, 15*(3), 288–297.
Moura, I. C., & van Hattum-Janssen, N. (2011). Teaching a CS introductory course: An active approach. *Computers & Education, 56*(2), 475–483.
Ng, W. (2012). Can we teach digital natives digital literacy? *Computers & Education, 59*(3), 1065–1078.
NRC (2012). *A framework for K-12 science education: Practices, crosscutting concepts, and core ideas*. The National Academies Press.
Olson, P. (2012). *Why Estonia has started teaching its first-graders to code*. In Forbes.
Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York: Basic Books.
Papert, S. (1994). *The children's machine: Rethinking school in the age of the computer*. Basic Books.
Pea, R. (1983). Logo programming and problem solving. In *American Educational Research Association*. Montreal, Canada.
Peppler, K. A., & Kafai, Y. (2007). From SuperGoo to Scratch: Exploring creative digital media production in informal learning. *Learning, Media and Technology, 32*(2), 149–166.
Polya, G. (1957). *How to solve it* (2nd ed.). Princeton, NJ: Princeton University Press.
Ratcliff, C. C., & Anderson, S. E. (2011). Reviving the turtle: Exploring the use of logo with students with mild disabilities. *Computers in the Schools, 28*(3), 241–255.
Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., et al. (2009). Scratch: Programming for all. *Communications of the ACM, 52*(11), 60–67.
Robertson, J. (2011). The educational affordances of blogs for self-directed learning. *Computers & Education, 57*(2), 1628–1644.
Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education, 13*(2), 137–172.
Sengupta, P., Kinnebrew, J., Basu, S., Biswas, G., & Clark, D. (2013). Integrating computational thinking with K-12 science education using agent-based computation: A theoretical framework. *Education and Information Technologies, 18*(2), 351–380.
Smith, D. C., Cypher, A., & Tesler, L. (2000). Novice programming comes of age. *Communications of the ACM, 43*(3), 75–81.
Søndergaard, H., & Mulder, R. A. (2012). Collaborative learning through formative peer review: Pedagogy, programs and potential. *Computer Science Education, 22*(4), 343–367.
Tangney, B., Oldham, E., Conneely, C., Barrett, S., & Lawlor, J. (2010). Pedagogy and processes for a computer programming outreach workshop – The bridge to college model. *IEEE Transactions on Education, 53*(1), 53–60.
Theodorou, C., & Kordaki, M. (2010). Super Mario: A collaborative game for the learning of variables in programming. *International Journal of Academic Research, 2*(4), 111–118.
Urquiza-Fuentes, J., & Velazquez-Iturbide, J. A. (2013). Toward the effective use of educational program animations: The roles of student's engagement and topic complexity. *Computers & Education, 67*, 178–192.
Utting, I., Cooper, S., Kölling, M., Maloney, J., & Resnick, M. (2010). Alice, greenfoot, and scratch – a discussion. *ACM Transactions on Computing Education (TOCE), 10*(4), 17.
Wang, L. C., & Chen, M. P. (2010). The effects of game strategy and preference-matching on flow experience and programming performance in game-based learning. *Innovations in Education and Teaching International, 47*(1), 39–52.
Wang, Y., Li, H., Feng, Y., Jiang, Y., & Liu, Y. (2012). Assessment of programming language learning based on peer code review model: Implementation and experience report. *Computers & Education, 59*(2), 412–422.
Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33–35.
Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A – Mathematical Physical and Engineering Sciences, 366*(1881), 3717–3725.
Wood, D., Bruner, J. S., & Ross, G. (1976). The role of tutoring in problem solving. *Journal of Child Psychology and Psychiatry, 17*(2), 89–100.
Yang, Y.-F. (2010). Students' reflection on online self-correction and peer review to improve writing. *Computers & Education, 55*(3), 1202–1210.
Yin, R. K. (2014). *Case study research: Designs and methods* (5th ed.). Thousand Oaks: SAGE Publications.
Zimmerman, B. J., & Tsikalas, K. E. (2005). Can computer-based Learning environments (CBLEs) Be used as self-regulatory tools to enhance learning? *Educational Psychologist, 40*(4), 267–271.